Unfold/fold Transformations in Definite Logic programs - A parameterized framework

Abhik Roychoudhury

K. Narayan Kumar

C.R. Ramakrishnan

I.V. Ramakrishnan

State University of New York at Stony Brook

A correct transformation sequence

```
p(X):-q(X).
q(a).
q(f(X)):-q(X).
```

Program P_0

```
p(a).
p(f(X)):-q(X).
q(a).
q(f(X)):-q(X).
```

Program P_1

```
p(a).
p(f(X)):-p(X).
q(a).
q(f(X)):-q(X).
```

Program P_2

Unfolding

Program P_i

$$
\begin{array}{l|c}
A{:-}\ A_1, A_2, ..., A_m & \\
B_1{:-}\ Body_1 & B_1\sigma_1 = A_1\sigma_1 \\
\dots & \dots \\
B_k{:-}\ Body_k & B_k\sigma_k = A_1\sigma_k
\end{array}
$$

Program P_{i+1}

$$(A{:-}\ Body_1, A_2, \dots, A_m)\sigma_1$$

$$\dots$$

$$(A{:-}\ Body_k, A_2, \dots, A_m)\sigma_k$$

Folding

Program P_j $(j \leq i)$	Program P_i
$D_1 \equiv \quad B_1 :- \ Body_1$	$C_1 \equiv \quad H :- \ Body_1\theta_1, A_2, ..., A_m$
$\ldots$	$\ldots$
$D_k \equiv \quad B_k :- \ Body_k$	$C_k \equiv \quad H :- \ Body_k\theta_k, A_2, ..., A_m$

where $B_1\theta_1 = \ldots = B_k\theta_k = B$ (say)

$D_1, \ldots, D_k$ are the only clauses of B in P_j (*folder clauses*)

certain conditions on $\theta_1, \ldots, \theta_k$ are satisfied.

Program P_{i+1}

$$C \equiv \quad H :- \ B, A_2, ..., A_m$$

Basic Correctness Issues

- **Semantics**: Least Herbrand Model semantics of definite logic programs.
- **Total Correctness**: A totally correct transformation sequence $P_0, P_1, \ldots, P_n$ of definite logic programs must have $M(P_n) = M(P_0)$
- All unfold/fold transformation sequences are not totally correct.

Program P_0	Program P_1
`p :- q.`	`p :- p.`
`q.`	`q.`

What is the basic problem ?

Folding may *introduce* circularity in definitions, thereby replacing finite proof paths with infinite ones.

Difficulty in proving total correctness:

To show total correctness of any sequence $P_0, \ldots, P_n$:
Need to prove: $M(P_i) \subseteq M(P_{i+1})$ and $M(P_i) \subseteq M(P_{i+1})$ for any unfold/fold step $P_i \rightarrow P_{i+1}$.

To show $M(P_i) \subseteq M(P_{i+1})$:
Need to prove: for all $A \in M(P_i)$ there is a proof of A in P_{i+1}
Attempt to prove: By induction on some well-founded order.

Program P_j	Program P_i	Program P_{i+1}
$B{:\!-}\ B_1, ..., B_k$	$A{:\!-}\ B_1, ..., B_k, A_2, ..., A_m$	$A{:\!-}\ B, A_2, ..., A_m$

To construct a proof of A in P_{i+1} we need a proof of B in P_{i+1}.

In general, this can only be shown by induction hypothesis. Then we must show $\alpha(A) \succ \alpha(B)$ where:

- α is some measure defined on ground atoms
- $\succ$ is a well-founded order over these mesures.

Observations:

- Simple well-founded order such as size of proof cannot be used.
- $\alpha(A)$ should be same for every program in a transformation sequence.

A solution

Attach abstract measures to program elements (atoms and clauses)

- Impose (mechanically checkable) conditions on *clause measures* to disallow incorrect folding steps.
- Induct on some ordering of *atom measures* to prove total correctness of any unfold/fold transformation sequence.

Clause measures :
For any transformation sequence $P_0, P_1, \ldots, P_n$ annotate each clause $C \in P_i$ with a pair $(\gamma^i_{lo}(C), \gamma^i_{hi}(C))$ where:

- $\gamma^i_{lo}(C), \gamma^i_{hi}(C) \in \mathcal{M}$ where $\langle \mathcal{M}, \oplus \rangle$ is a commutative group and $\oplus$ is montonic w.r.t a linear order $\succ$
- $\gamma^i_{hi}(C) \succeq \gamma^i_{lo}(C)$

General Intuition: In a concrete unfold/fold transformation system

- Clause measures $\gamma^i_{lo}(C), \gamma^i_{hi}(C)$ denote the minimum and maximum savings (in proof size) incurred by applying C in a ground proof in P_i.
- Roughly speaking, a folding step is allowed when the previously accumulated savings are not offset by this folding step.[a]

[a]This does not mean that we must unfold before folding.

Atom measures: $\forall A \in M(P_0)$ attach an atom measure $\alpha(A) \in \mathcal{W}$ where $\mathcal{W}$ is a subset of $\mathcal{M}$ over which $\succ$ is well-founded.

General Intuition: In a concrete unfold/fold transformation system

- Atom measure $\alpha(A)$ is indicative of the "weight" of a *canonical* (shortest) proof of A in P_0.
- For any clause $C \equiv H :- B_1, \ldots, B_n$ applied in a canonical proof in P_i the clause measures of C bound the difference in the atom measures of the head and the body, *i.e.*

$$\gamma^i_{lo}(C) \preceq \alpha(H\theta) - \sum_{1 \leq l \leq n} \alpha(B_l\theta) \preceq \gamma^i_{hi}(C)$$

Program P_j	Program P_i
$D \equiv B\text{:-}\ B_1, ..., B_k$	$C \equiv A\text{:-}\ B_1, ..., B_k, A_2, ..., A_m$

Then, assuming the existence of canonical proofs in $P_0, \ldots, P_i$

$$\alpha(B) \preceq \gamma^i_{hi}(D) \oplus \textstyle\sum_{1 \le l \le k} \alpha(B_l)$$
$$\alpha(A) \succeq \gamma^i_{lo}(C) \oplus \textstyle\sum_{1 \le l \le k} \alpha(B_l) \oplus \sum_{2 \le l \le m} \alpha(A_l)$$

Since $\alpha(A_l) \succeq \mathbf{0}$, then if $\gamma^i_{lo}(C) \succ \gamma^i_{hi}(D)$ we have $\alpha(A) \succ \alpha(B)$

Measure Consistency For any program P_i in a transformation sequence $P_0, \ldots, P_n$, we also need to prove the existence of a canonical proof of A for every $A \in M(P_i)$. This abstract property is called *Measure Consistency*.

Measure-preserving unfolding

Program P_i

$A{:}\!-\ A_1, A_2, ..., A_m \quad (\gamma, \gamma')$		
$B_1{:}\!-\ Body_1 \quad (\gamma_1, \gamma_1')$		$B_1\sigma_1 = A_1\sigma_1$
$\ldots$		$\ldots$
$B_k{:}\!-\ Body_k \quad (\gamma_k, \gamma_k')$		$B_k\sigma_k = A_1\sigma_k$

Program P_{i+1}

$$(A{:}\!-\ Body_1, A_2, \ldots, A_m)\sigma_1 \quad (\gamma \oplus \gamma_1, \gamma' \oplus \gamma_1')$$

$$\ldots$$

$$(A{:}\!-\ Body_k, A_2, \ldots, A_m)\sigma_k \quad (\gamma \oplus \gamma_k, \gamma' \oplus \gamma_k')$$

Measure preserving Folding

Program P_j $(j \leq i)$	Program P_i
$B_1 \text{:-}\ Body_1 \quad (\delta_1, \delta'_1)$	$H \text{:-}\ Body_1\theta_1, A_2, ..., A_m \quad (\gamma_1, \gamma'_1)$
$\ldots$	$\ldots$
$B_k \text{:-}\ Body_k \quad (\delta_k, \delta'_k)$	$H \text{:-}\ Body_k\theta_k, A_2, ..., A_m \quad (\gamma_k, \gamma'_k)$

If: $B_1\theta_1 = \ldots = B_k\theta_k = B$ (say)

$\gamma_1 \succ \delta'_1, \gamma_2 \succ \delta'_2, \ldots, \gamma_m \succ \delta'_m$

all other conditions of folding are satisfied.

Program P_{i+1}

$$H \text{:-}\ B, A_2, ..., A_m \quad (min_{1 \leq l \leq m}(\gamma_l \ominus \delta'_l), max_{1 \leq l \leq m}(\gamma'_l \ominus \delta_l))$$

Why approximate clause measures?

Clause measures denote the savings in proof size by applying that clause in a ground proof.

Exact savings cannot be estimated for clauses introduced by disjunctive folding.

Program P_i	Program P_{i+1}
C_1: `p :- r, t.` (x_1)	C': `p :- q, t.` $(?)$
C_2: `p :- s, t.` (x_2)	C_3: `q :- r.` (x_3)
C_3: `q :- r.` (x_3)	C_4: `q :- s.` (x_4)
C_4: `q :- s.` (x_4)	

The savings by using C' in a proof depends on whether C_3 or C_4 is used to resolve `q` in that proof.

State of the Art

- *Tamaki-Sato '84*
  Introduced totally correct unfold/fold transformations in logic programs. Folding uses a single non-recursive clause as folder.

- *Tamaki-Sato '86*
  Folding uses only one clause as folder (conjunctive folding).

- *Kanamori-Fujita '87*
  A folding step is conjunctive. Introduced the idea of annotating clauses with integer counters to maintain correctness.

- *Gergatsoulis-Katzouraki '94*
  A folding step can use multiple non-recursive clauses as folder.

Advantages of our approach

- Unlike existing systems, it does not impose syntactic restrictions on the folder clauses in a folding step. Folding is restricted only by conditions on the clause measures.

- Allows us to identify an abstract program property (*Measure Consistency*) which ensures correctness of unfold/fold systems. We can intrdouce new transformations (*e.g. Goal Replacement*) without redoing the correctness proofs.

- Existing unfold/fold transformation systems are specific instances of our framework. This allows us to understand the strengths and limitations of existing systems.

- Construction of a new unfold/fold transformation system
  - The predicate symbols are a-priori partitioned into n strata.
  - The atom and clause measures are n-tuples of integers.
  - Addition ($\oplus$) of the measures correspond to coordinatewise addition, and comparison ($\succ$) corresponds to lexicographic ordering of n-tuples.

Note:
1) This transformation system poses no syntactic restriction on the folder clauses.
2) It can be shown to be more general than existing systems (in terms of transformation sequences allowed).

Application

Verifying *temporal properties* of *parameterized concurrent systems* (infinite families of finite-state concurrent systems) using unfold/fold transformations.

These problems can be encoded as a logic program as follows:

- `gen`: Encodes definition of the parameterized system.
- `test`: Encodes temporal property to be verified.
- `trans`: Encode the transition relation of the members of the infinite family
- `thm`: Which members of the infinite family satisfy the property

```
thm(X) :- gen(X), test(X).
```

A typical example

Initial Program:

```
thm(X) :- gen(X), test(X).
gen([]).
gen([0|X]) :- gen(X).
canon([]).
canon([1|X]) :- canon(X).
test(X) :- canon(X).
test(X) :- trans(X,Y), test(Y).
trans([0|X], [1|X]).
trans([1|T],[1|T1]) :- trans(T,T1).
```

Transformed Program:

```
thm([]).
thm([0|X]) :- thm(X).
```

Example Problems :

- verifying liveness in a n-bit shift register for any n
- verifying mutual exclusion in a n-process token ring for any n
- verifying liveness in the family of binary trees

These problems exploit the power of:

- our more general folding rule, which allows folding with multiple recursive folder clauses
- stratification and counting in the clause measures

Extensions and Future work

- This work has been extended to obtain a similar generalized unfold/fold transformation system for normal logic programs [ASIAN99]. Uniform proof of correctness w.r.t. different semantics of negation : well-founded model, stable model, partial stable model.

- Can we develop similar parameterized transformation framework for other programming paradigms[a], *e.g.*

  – functional languages (current work [Sands96])

  – process algebraic languages (current work [FS98])

[a] Efforts in this direction: Proietti and Pettorossi - ICLP '99